A MODULAR, SCALABLE, ARCHITECTURE FOR UNMANNED VEHICLES

David G. Armstrong II, Carl D. Crane III, David Novick, Jeffrey Wit
Center for Intelligent Machines and Robotics
University of Florida, Gainesville, Florida 32611, USA
352-392-0814, 352-392-1071 (fax)
ccrane@ufl.edu, http://www.me.ufl.edu/CIMAR

Ralph English
Wintec, Inc.
Ft. Walton Beach, Florida

Phillip Adsit
Applied Research Associates
Tyndall Air Force Base, Florida

Cpt. David Shahady
Air Force Research Laboratory
Tyndall Air Force Base, Florida

ABSTRACT: A modular, scalable architecture for use on unmanned vehicles has been developed at the Center for Intelligent Machines and Robotics under the direction of the Air Force Research Laboratory at Tyndall Air Force Base, FL. This state of the art architecture isolates five functionally cohesive sub-tasks into self-contained modules with well defined interfaces. The architecture consists of a Mobility Control Unit (MCU), Path Planner (PLN), Position System (POS), Detection and Mapping System (DMS), and a Primitive Driver (PD). The design considerations for the development of this architecture included sub-system modularity, implementation independent software interfaces, ability to expand system functionality through continued addition of modules (scale), and the goal of moving toward a standard architecture for autonomous systems. The focus of this paper is to present a modular architecture that addresses the above design considerations.

One particularly noteworthy aspect of the architecture is the use of propulsive and resistive wrench commands that are communicated to the Primitive Driver component to specify desired vehicle motion. The advantage of this approach is that a standard command can be used to communicate with a wide variety of vehicles such as steered-wheeled, tracked, or omni-directional. It is not necessary to have vehicle specific commands included in the architecture such as 'steering wheel angle', 'throttle position', 'left-track velocity', and 'right-track velocity'. Without the use of the propulsive and resistive wrench commands, the number of needed commands was potentially limitless with only a small subset of commands actually be applicable to a particular vehicle.

The architecture has been validated by implementing it on three mobile platforms, i.e. a steered wheel vehicle, a tracked vehicle, and an omni-directional indoor vehicle. Several features of this work have been incorporated in the architecture that is being adopted by the Joint Architecture for Unmanned Ground Systems (JAUGS) Working Group.

Keywords: Architectures, Autonomous Vehicles, Navigation, Modular, Intelligent

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **2000**	2. REPORT TYPE	3. DATES COVERED **00-00-2000 to 00-00-2000**
4. TITLE AND SUBTITLE **A Modular, Scalable, Architecture for Unmanned Vehicles**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Center for Intelligent Machines and Robotics,Department of Mechanical Engineering,University of Florida,Gainesville,FL,32611**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES **The original document contains color images.**		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**15**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

1. INTRODUCTION

The Center for Intelligent Machines and Robotics (CIMAR), at the University of Florida, has worked to develop a series of autonomously navigating vehicles. This work has been sponsored by the Air Force Research Laboratory at Tyndall Air Force Base, Florida. A Kawasaki Mule 500 all-terrain vehicle, named the Navigation Test Vehicle, was first modified for computer control in 1991 for the purpose of providing a test and development platform (see figure 1.). The technology developed on this platform has since been applied to several vehicles including heavy construction equipment.

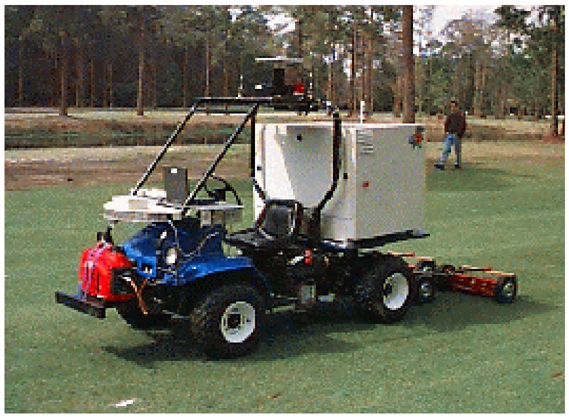
Figure 1: Navigation Test Vehicle

The original vehicle control architecture was primarily based on a shared memory (blackboard) approach, implemented through the use of multiple 68030 CPU boards running on a VME backplane. The use of shared memory provided the advantage of running critical real-time procedures in parallel and having their resultant data available to all other programs immediately via the VME backplane. This is the key advantage of using shared memory.

The problem with shared memory is that it tightly couples all of the sub-systems in an indirect way making programming errors in the system difficult to trace. The shared or common memory area becomes unmanageable in that a piece of data can be over written in error with impact somewhere else in the system. The result is a system that becomes difficult to maintain or upgrade as new features and hardware are added. In addition, the integration of all the systems into one backplane makes it difficult to use any one sub-system on a different project. For example to apply the position system to another autonomous vehicle would most certainty require significant changes to both hardware and software.

From the experience gathered over the past years of work, a list of four architecture design requirements has emerged. The architecture should:

1. be comprised of a set of well defined, self contained sub-modules
2. exhibit implementation independent well defined software interfaces
3. have the ability to scale up a system's functionality with different combinations of modules
4. provide a stepping stone toward the development of a standard architecture

The focus of this paper is to present a state-of-the-art modular architecture which addresses the above design considerations.

2. ARCHITECTURE

The task of autonomous navigation can be broken down into five sub-tasks as follows:

1. Vehicle positioning
2. Vehicle Specific Actuator Control
3. Path Planning
4. Perception of the Environment
5. Path Execution and Obstacle Avoidance

The proposed architecture isolates these five, functionally cohesive, sub-tasks into self contained modules with well defined interfaces. The architecture, depicted in Figure 2, consists of a Position System (POS), Primitive Driver (PD), Path Planner (PLN), Detection and Mapping System (DMS), and an Autonomous Control Unit (ACU).

The most important part of defining an architecture is specifying the messaging or interfaces between componets. The interface defines what information goes in and what information comes out, thereby indirectly specifying the function of the component. It is important to note that the interface does not and should not specify how the function is carried out. This allows the designer to be flexible in the choice of the hardware and software to best suit the module's functions. This paper will now proceed with a brief discussion of each component including its function and the associated software interface.

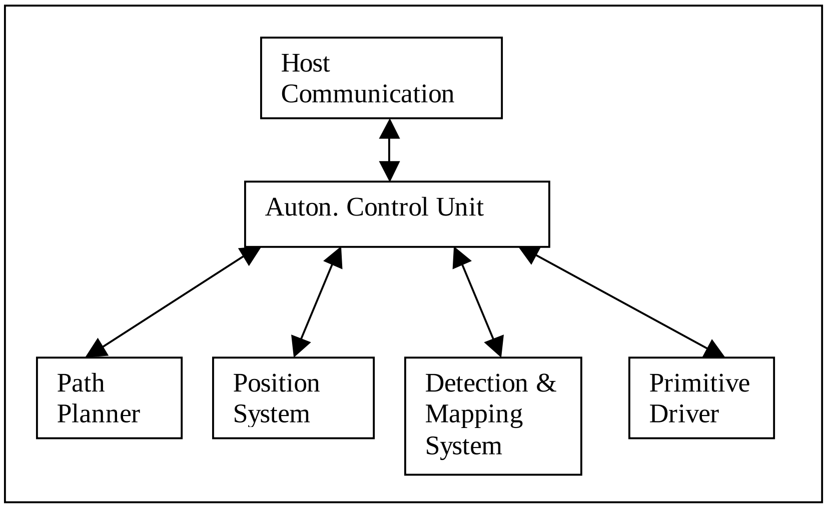

Figure 2: Modular and Scalable Architecture

2.1 Position System (POS)

The function of this module is to provide position and orientation data to the Autonomous Control Unit. The position system, like all of the modules, exists as a stand alone, self-contained unit. The interface has been developed in a generic manner (independent of its implementation) so that when new technology becomes available it can be plugged in without changing the host system's software.

The contents of the position message consist of a set of generic parameters including latitude, longitude, elevation, roll, pitch, and yaw. These parameters are sufficient to completely describe the vehicle's pose. Just as important however, is the absence of any implementation specific data such as GPS or INS specific parameters that would only be useful for a system that was based on that type of implementation. It is also important to note that the position system interface reports the position data in only one single format. Since the global coordinates are sufficient to describe any possible vehicle pose it is not necessary to support a separate set of pose data to describe a relative position. Providing for more than one form of data output from any given module can lead to exponential complexity as data flows upward and outward throughout the system.

One example of a position system that has been implemented includes an inertial navigation system (INS) and a differential global position system (DGPS). The INS offers position data at a 10Hz rate but tends to drift over time. The DGPS provides position data without drift but only at a 1Hz rate and is subject to availability of the orbiting satellites. Through the application of a Kalman Filter, the system is able to maintain accuracies equal to that of the DGPS and an output rate of 10Hz while smoothing through temporary loss of DGPS signals (see Wit, et al., 1996). An average position accuracy of 6cm is obtained with a standard deviation of 3cm. A summary of the messaging interface for the Position System is listed in Table 1.

2.2 Primitive Driver (PD)

The Primitive Driver accepts higher level commands from the Autonomous Control Unit that describe how the vehicle is to move. It then translates these commands into the low-level commands that directly control the vehicle actuators to achieve the desired motion. Only those vehicle actuators that are directly related to vehicle mobility are controlled by the Primitive Driver. A summary of the messaging interface for the Primitive Driver is listed in Table 2.

A significant feature of the Primitive Driver system is the fact that mobility commands are specified by only two wrench commands, i.e. a propulsive wrench and resistive wrench. The propulsive wrench specifies how the vehicle should move while the resistive wrench specifies how the vehicle should act to impede movement.

Figure 3 shows a vehicle with a coordinate system attached. In this example the X axis is in the forward direction of travel, the Z axis is downward, and the Y axis is defined based on a right handed coordinate system. It is assumed that the origin of this coordinate system is located at the vehicle center of mass. It is important to point out however that exact location of the center of mass of the vehicle will not be required.

The six coordinates of the propulsive wrench can be written as

$$\hat{\mathbf{w}}_p = \{f_{px}, f_{py}, f_{pz} ; m_{px}, m_{py}, m_{pz}\} \tag{1}$$

while the six coordinates of the resistive wrench can be written as

$$\hat{\mathbf{w}}_r = \{f_{rx}, f_{ry}, f_{rz} ; m_{rx}, m_{ry}, m_{rz}\} \ . \tag{2}$$

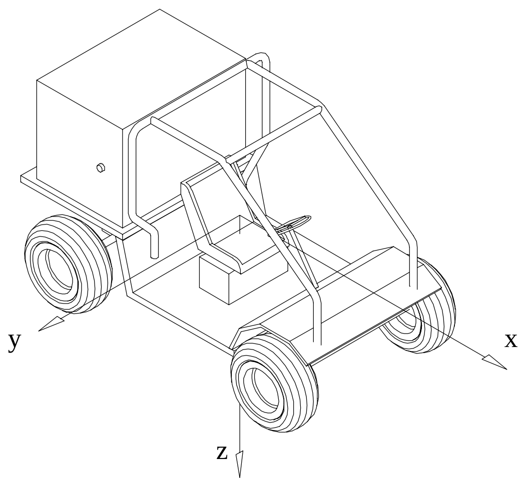

Figure 3: Vehicle Coordinate System

Since the reference point is assumed to be located at the vehicle center of mass, the force components of the propulsive wrench specify a net force that is to be applied to the vehicle in order to cause it to translate. The moment components specify a net moment that will cause the vehicle to rotate or change orientation. For the resistive wrench, the force and moment components specify how the vehicle is to act to resist motion. When a vehicle system receives the propulsive and resistive wrench commands, the vehicle actuators must act in a way as to apply appropriate translational and rotational propulsion or resistance. This approach represents a general means of implementing vehicle control commands.

The significant feature of this approach is to avoid having to have different motion commands for different vehicles, such as steering and throttle commands for a steered-wheeled vehicle, and left track / right track commands for a tracked vehicle. Without a generic definition for desired motion it would be necessary for the the architecture to have a plethora of vehicle commands such as 'desired steering wheel position', 'desired throttle position', 'desired left track velocity', 'desired right track velocity', and 'desired brake setting' to name a few. The number of commands that would have to be included in the architecture documentation potentially was limitless since new commands would have to be added for each new mobility concept. Only a small number of these commands would be relevant to any one particular vehicle.

2.3 Path Planner (PLN)

The function of this module is to obtain a sequence of via-points or sub-goals (See Jarvis, 1983; Rankin, et al., 1996). These sub-goals will direct the robot from a specified start pose to a specified goal pose in a manner as to avoid collision with all known obstacles while minimizing the total distance traveled. A summary of the messaging interface for the planner is listed in Table 3.

2.4 Detection and Mapping System (DMS)

The Detection and Mapping System, like the others, is developed as a stand alone plug-in module with a hardware independent interface. The function of this module is to build and maintain a representation of the environment (Area Map) and to make information contained in the Area Map available to the host. The DMS report consists of changes to the Area Map. When the DMS is given a request Area Map, it responds with a message that includes every change to the Area Map since the specified time. If the time is given as zero, then the DMS will transmit the entire Area Map. After the first report is sent, the DMS will continue to send Area Map reports at the rate specified in the Request Area Map message. The Area Maps that are sent after the first one consist only of updates since the last report (includes obstacles added or deleted).

The DMS that has been implemented combines data from both ultrasonic and stereo vision sensors. An array of 24 ultrasonic transducers has been utilized to provide range data up to 7 meters at approximately 3Hz. In addition, an obstacle detection system based on stereo vision (a system built by the NASA Jet Propulsion Laboratory) has been used to provide data at longer ranges (up to 30) meters at approximately 2Hz (see Matthies, et al., 1996). Data from both obstacle detection systems are fused together and shipped out to the host system in the form of an Area Map. A summary of the messaging interface for the DMS is listed in Table 4.

2.5 Autonomous Control Unit (ACU)

The Autonomous Control Unit (ACU) can be thought of as the integrator of the sub-modules and is the higher order agent that achieves mobility. The function of the ACU is to respond to high-level commands from the host (typically an Operator Control Unit) and coordinate the sub-modules to achieve autonomous path execution. This includes communication with all sub-modules and moving the vehicle along the planned path while avoiding both expected and unexpected obstacles. The sub-tasks of the ACU include the following:

- Process commands from the host
- Read input from the Position Module and the Detection and Mapping System
- Request the Path Planner to plan a path to the goal
- Maintain the current sub-goal in following the path
- Perform path execution based on combined proportional/integral (PI) and pure pursuit control methods
- Perform reflexive obstacle avoidance based on the Area Map provided in real time from the DMS
- Perform path re-planning (using the Planner Module) to avoid “long range” obstacles
- Output the propulsive and resistive wrench commands to the Primitive Driver module

A summary of the messaging interface for the Autonomous Control Unit is listed in Table 5.

3. IMPLEMENTATION AND CONCLUSIONS

A modular, scalable architecture has been presented. This architecture has been implemented on the Navigation Test Vehicle shown in Figure 4 as well as the All-Purpose Remote Transport System (ARTS) and a Cybermotion omni-directional robot shown in Figures 5 and 6. The experience of applying the modular architecture to the ARTS demonstrated the

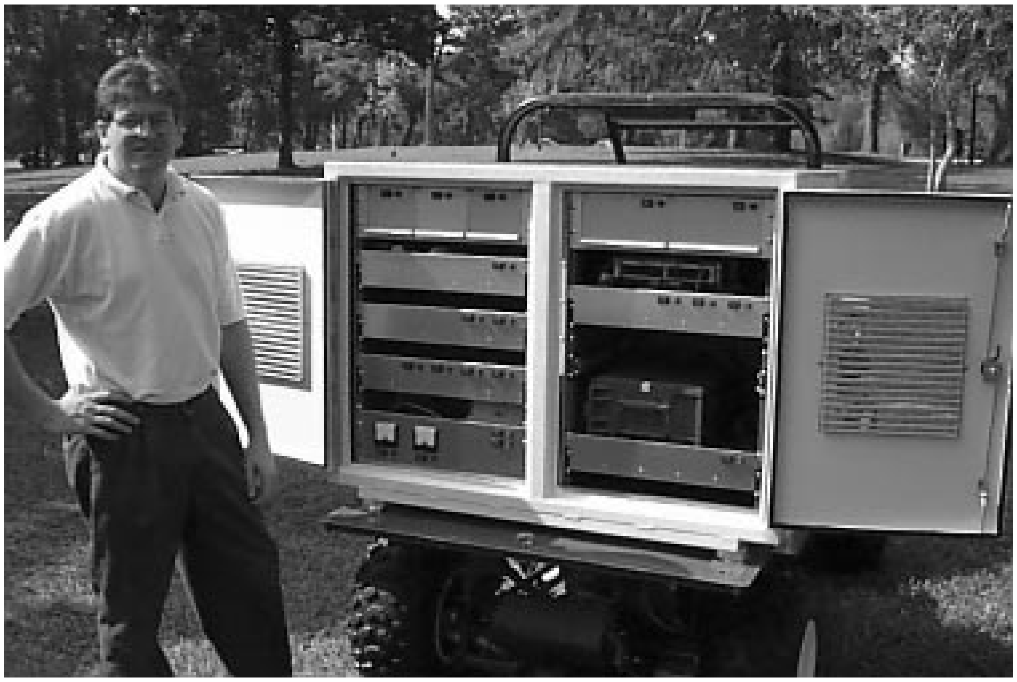

Figure 4: Navigation Test Vehicle with Modular System Installed

advantages of the modular approach. Two teams of developers were able to implement autonomous navigation of the ARTS in a period of six weeks from the start of the project. One group at Tyndall Air Force Base designed and implemented the Primitive Driver component on the ARTS so that the system would respond appropriately to propulsive and resistive wrench commands. The second group assembled ACU and POS modules that were duplicates of the

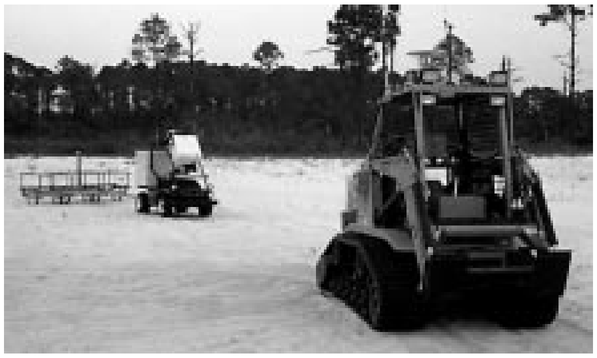

Figure 5: NTV and ARTS Autonomous Vehicles

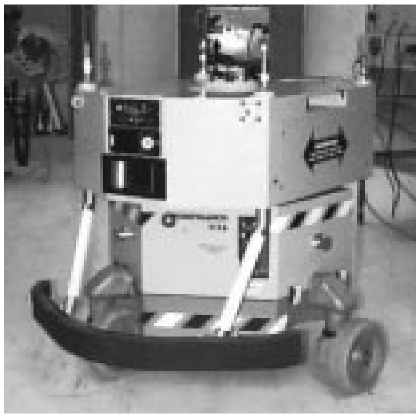

Figure 6: Cybermotion Robot

same modules running on the NTV. When the duplicate modules were installed on the ARTS, the team was able to effect autonomous navigation within twenty minutes. Only one software modification had to be made during the integration process. This was to improve the path tracking accuracy of the ARTS by changing control tuning parameters on the ACU that govern how the propulsive and wrench commands are generated in response to path tracking errors

This implementation serves to demonstrate the advantages and flexibility of a modular hardware architecture that is integrated via a standard messaging protocol. This approach will allow new hardware and sensor systems to be added to any autonomous vehicle so that it may be configured and made operational in a minimal amount of time.

4. REFERENCES

Jarvis, R.A., "Growing Polyhedral Obstacles for Planning Collision-Free Paths," The Australian Computer Journal, 15(3), 1983, pp. 103-111.

Rankin, A., and Crane, C., "Multi-Purpose Off-Line Path Planning Based on an A* Search Algorithm,"Proceedings of the 1996 ASME Mechanisms Conference, Irvine, Ca., Published on CD-ROM.

Wit, J.S., Crane, C.D., Armstrong, D.G., "Evaluation of an Integrated Inertial Navigation System and Global Positioning System Under Less Than Optimal Conditions," Proceedings of the Sixth International Symposium on Robotics and Manufacturing (ISRAM '96), Volume 6, ASME Press, Montpellier, France, May 1996, pp. 649-654.

Matthies, L., Brown, E.. "Machine Vision for Obstacle Detection and Ordnance Recognition," Annual meeting of the Association for Unmanned Vehicle Systems (AUVSI'96), Orlando, FL, July 1996.

5. ACKNOWLEDGEMENTS

The authors would like to gratefully acknowledge the support and encouragement of personnel of the Air Force Research Laboratory at Tyndall Air Force Base, Florida.

Table 1: Position System Messaging Interface

Input Commands

Reinitialize:	Commands the Position System to reinitialize each sub-system in the proper sequence to bring the system up to a state of readiness. The Position System must be initialized for the Position Message to be valid (with the exception of the two status bytes which are always valid).
Standby:	Commands the Position System to place each of its sub-systems in a state where they are ready to be initialized.
Set Config.:	This command is used to set the configuration of the Position System
Shutdown:	Commands each of the sub-systems to shutdown in the proper sequence.

Input Request

- Request Position
- Request Status
- Request Configuration

Output Messages

Position:	Latitude	Theta X	Velocity X	Omega X	Time Stamp
	Longitude	Theta Y	Velocity Y	Omega Y	Status Bytes
	Elevation	Theta Z	Velocity Z	Omega Z	Data Validity
	Position RMS	Attitude RMS	Velocity RMS	Omega RMS	

Status:	Startup:	Indicates the system has just been powered up
	Busy:	Indicates the system is currently processing the last command
	Standby:	Indicates the following statements apply: - The system is ready to be reinitialize - The Position Message is not valid (with the exception of the two status bytes that are always valid)
	Ready:	Indicates that the system is initialized and is operational
	Problem:	Indicates that a self-correcting problem has occurred and the problem is being corrected internally. This problem requires no input from the host
	Error:	Indicates that a problem has occurred that the system could not resolve. An error requires the intervention of the host to be resolved.
	Failure:	Indicates that the system has failed and will not recover.
	Note: Each of the five modules uses this same status layout	
Configuration:	Text Description of the system & component identification Reference Latitude, Longitude, and Elevation	

Table 2: Primitive Driver Messaging Interface

Input Commands					
	Reinitialize:	Commands the Primitive Driver to reinitialize each sub-system in the proper sequence to bring the system up to a state of readiness. Once initialized, the Primitive Driver will respond to commands that cause or resist vehicle motion. The Primitive Driver must be initialized for vehicle motion to occur.			
	Standby:	Commands the Primitive Driver to place each of its sub-systems in a state where they are ready to be initialized.			
	Propulsive Wrench	This command, along with the Resistive Wrench Command, tells the vehicle how to move. A wrench is Force-Moment vector made up of six components as shown below:			
		% Force X	% Moment X		
		% Force Y	% Moment Y		
		% Force Z	% Moment Z		
		The propulsive Wrench is applied to the center of mass point of the vehicle and is used to propel the vehicle. The force component of the wrench acts to translate the vehicle while the moment component acts to rotate the vehicle. Essentially we are telling the Primitive Driver how we want to push on the vehicle were the percentage indicates the magnitude of the push. For example: if the vehicle were a car, then the Primitive Driver would map a 50% Force X to the throttle and transmission (50% throttle and transmission in drive) and Moment Z would map directly to the steering. The remaining parameters would not be used.			
	Resistive Wrench	This command is similar to the Propulsive Wrench however the Resistive Wrench acts to impede vehicle motion. The resistive Wrench uses the same six parameters as the Propulsive Wrench. For example, if the vehicle were a car, then the a 20% Force X command would map directly to the brake (brake depressed 20%).			
	Shutdown:	Commands each of the sub-systems to shutdown in the proper sequence.			
Input Request					
	Request Status				
	Request Config.				
Output Messages					
	Status:	Startup:	Indicates the system has just been powered up		
		Busy:	Indicates the system is currently processing the last command		
		Standby:	Indicates the following statements apply:		
			- The system is ready to be reinitialize		
			- The system will not respond to commands that cause or resist motion		
			- The vehicle should remain stationary		
			- The vehicle actuators should not move		
			- From a mobility standpoint, the vehicle should be considered safe		
		Ready:	Indicates that the system is initialized and is operational		
		Problem:	Indicates that a self-correcting problem has occurred and the problem is being corrected internally. This problem requires no input from the host		
		Error:	Indicates that a problem has occurred that the system could not resolve. An error requires the intervention of the host to be resolved.		
		Failure:	Indicates that the system has failed and will not recover.		
	Configuration:	Text Description of the system & component identification			
		Vehicle Length	Max Speed X	Max Speed -X	Max Omega X
		Vehicle Width	Max Speed Y	Max Speed -Y	Max Omega Y
		Vehicle Height	Max Speed Z	Max Speed -Z	Max Omega Z
		Turning Radius	Max Theta X	Max Theta Y	

Table 3: Path Planner Messaging Interface

Input Commands		
Reinitialize:		Commands the Path Planner to reinitialize to bring the system up to a state of readiness. This may include updating the systems map database and performing some preprocessing of that information.
Set Config.:		This command is used to set the configuration of the Path Planner.
Shutdown:		Commands each of the sub-systems to shutdown in the proper sequence.
Input Request		
Request Path		This request would include such information such as the start and goal positions if a "go-to-goal" plan is desired or the corner points of a bounded area if a "sweep plan" is desired. In either case the latitude, longitude, elevation, roll, pitch, and yaw of each point would be specified.
Request Status		
Request Config.		
Output Messages		
Planned Path:		Includes the latitude, longitude, elevation, roll, pitch, and yaw of each point in the path In addition, information such as the path type, path length, number of sub-goals, and path status are reported.
Status:	Startup:	Indicates the system has just been powered up
	Busy:	Indicates the system is currently processing the last command
	Standby:	Indicates the system is ready to be reinitialized
	Ready:	Indicates that the system is initialized and is operational
	Problem:	Indicates that a self-correcting problem has occurred and the problem is being corrected internally. This problem requires no input from the host
	Error:	Indicates that a problem has occurred that the system could not resolve. An error requires the intervention of the host to be resolved.
	Failure:	Indicates that the system has failed and will not recover.
Configuration:		Includes a text description of the system as well as the vehicle length, width, height, and turning radius.

The Path Planner system may also request a World Modeling System Area Map Report as defined by the World Modeling System.

Table 4: Detection and Mapping System Messaging Interface

Input Commands			
	Reinitialize:	Commands the DMS to reinitialize each sub-system in the proper sequence to bring the system up to a state of readiness. Once initialized, the DMS will respond to request for the Area Map.	
	Standby:	Commands the Detection and Mapping System to place each of its sub-systems in a state where they are ready to be initialized.	
	Shutdown:	Commands each of the sub-systems to shutdown in the proper sequence.	
Input Request			
	Request Area Map		
	Request Status		
	Request Config.		
Output Messages			
	Area Map	This message would include the number of obstacles, addition or deletion, ID#, confidence, time stamp, classification (tree), type (oak), class & type confidence, and the vertices of each obstacle in global coordinates.	
	Status:	Startup:	Indicates the system has just been powered up
		Busy:	Indicates the system is currently processing the last command
		Standby:	Indicates the system is ready to be reinitialized
		Ready:	Indicates that the system is initialized and is operational
		Problem:	Indicates that a self-correcting problem has occurred and the problem is being corrected internally. This problem requires no input from the host
		Error:	Indicates that a problem has occurred that the system could not resolve. An error requires the intervention of the host to be resolved.
		Failure:	Indicates that the system has failed and will not recover.
	Configuration:	Text Description of the system & component identification	

The Detection and Mapping System may also request a Position Message as defined by the Position System. The DMS would use this information to report all Map coordinates in a global sense.

Table 5: Autonomous Control Unit Messaging Interface

Input Commands		
	General:	All of the input and output messages from the other systems may pass through the ACU. For example, the host may request a position message from the ACU. The ACU would then forward this message to the Position System and subsequently forward the Position Message back to the host.
	Reinitialize:	Commands the ACU to reinitialize itself to bring the system up to a state of readiness. The ACU will also verify that the Position System, Path Planner, Primitive Driver, and World Modeling System are all properly initialized.
	Standby:	Commands the ACU to place all of its sub-systems in a state where they are ready to be initialized. A standby command will be sent to the Position System, Path Planner, Primitive Driver, and World Modeling System.
	Execute Path:	Commands the ACU to begin moving along the planned path. The planned path is either sent with this message or the currently loaded path is used.
	Pause:	Commands the vehicle to stop path execution. The ACU's sub-systems will remain in the active (ready) state so that path execution can continue immediately upon receipt of the ACU's continue message. If a more secure, stopped, state is desired than the Standby message should be used as this will place all of the ACU's sub-systems in a safe (Standby) mode.
	Continue:	Continues path execution from the point that it had previously left off.
	Set Mode:	Sets the mode of operation such as safe, autonomous, teleop, or teach mode.
	Shutdown:	Commands each of the sub-systems to shutdown in the proper sequence.
Output Messages		
	Startup:	Indicates the system has just been powered up
	Busy:	Indicates the system is currently processing the last command
	Standby:	Indicates the system is ready to be reinitialize
	Ready:	Indicates that the system is initialized and is operational
	Problem:	Indicates that a self-correcting problem has occurred and the problem is being corrected internally. This problem requires no input from the host
	Error:	Indicates that a problem has occurred that the system could not resolve. An error requires the intervention of the host to be resolved.
	Failure:	Indicates that the system has failed and will not recover.